Programming Fundamental

Week 13

Learning Outcomes:

After this lesson, students will be able to:

1. To declare pointer variable
2. Access address of memory location
3. Passing array into function as an argument
4. Passing address of memory location as an argument.
5. Declaring dynamic memory location.

Instructions

- Use proper indentation to make your programs readable.
- Use descriptive variables in your programs (Name of the variables should show their purposes)

Pointer

It is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is −

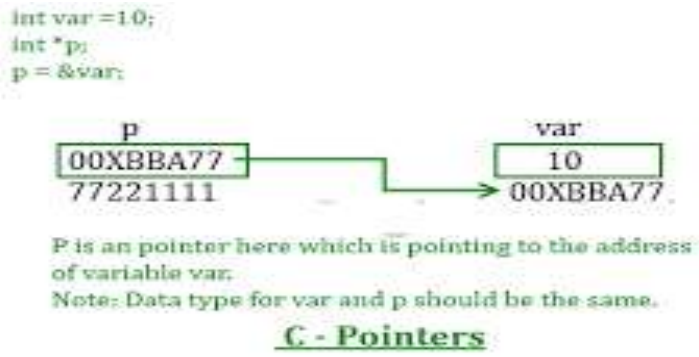
type *var-name;

Note: * is called indirection operator

Declaration:

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

Initializing Pointer:



Example #1:

Write a program that shows address of a variable using & operator.

Solution

`#include <iostream>`
`using namespace std;`
`int main()`
`{`
`  int var = 5;`
`  cout<<"var:"<<var;`
 	
`  cout<<"address of var:"<<&var;`
`  return 0;`
`}`

The code produces the following output

```
var:5
address of var:0x6ffe0c
--------------------------------
```

Example #2:

Write a program that assigns address of integer variable to pointer variable using & operator.

Solution

```
#include <iostream>
using namespace std;
int main()
{
 int* pc;
 int  c;
c = 5;
pc = &c;
cout<<"address of C="<<&c<<endl;
cout<<"Pc="<<pc<<endl;
}
```

The code produces the following output

```
address of C=0x6ffe14
Pc=0x6ffe14
```

Example #3:

Write a program that assign address to pointer variable and use that pointer variable to show the value of that address.

Solution

```
#include <iostream>
using namespace std;
int main()
{
 int* pc;
 int  c;
c = 5;
pc = &c;
cout<<"address of C="<<&c<<endl;
cout<<"Pc="<<*pc<<endl;
}
```

The code produces the following output

```
address of C=0x6ffe04
Pc=5
```

Example#4

Write a program that assigns address of an integer to pointer variable and change the value of integer variable and access the updated value using pointer.

Solution

```
#include <iostream>
using namespace std;
int main()
{
 int* pc, c;
c = 5;
pc = &c;
c=1;
cout<<"value of C="<<c<<endl;
cout<<"Pc="<<*pc<<endl;
}
```

The code produces the following output

```
value of C=1
Pc=1
```

Example#5

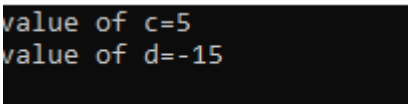
Write a program that shows the behavior of a pointer , like pointer pc show value of c if you assign it address c and same pointer pc show value of d if assign address of d.

Solution

```
#include <iostream>
using namespace std;
int main()
{
 int* pc, c, d;
c = 5;
d = -15;

pc = &c;
cout<<"value of c"<<*pc;
pc = &d;
 cout<<"value of d"<<*pc;
}
```

The code produces the following output



Call by value in C++

- In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
- In call by value method, we cannot modify the value of the actual parameter by the formal parameter.
- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Example#6

Make a function named as change with one integer argument. Update the value of argument and return to main body. Display the value of passing argument again.

Solution

```
#include<iostream>
using namespace std;
void change(int num) {
    cout<<"Before adding value inside function num="<<num<<endl;
    num=num+100;
     cout<<"After adding value inside function num="<<num<<endl;
}
int main() {
    int x=100;
    cout<<"Before function call x="<< x<<endl;
    change(x);//passing value in function
    cout<<"After function call x="<< x<<endl;
return 0;
}
```

The code produces the following output

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=100
```

Call by reference

- In call by reference, the address of the variable is passed into the function call as the actual parameter.

- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Example#7 Make a function named as change with one integer argument using address operator & . Update the value of argument and return to main body. Display the value of passing argument again.

Solution

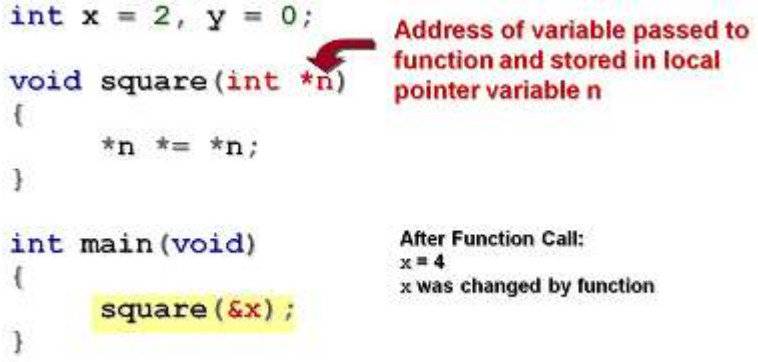
```
#include<iostream>
using namespace std;
void change(int &num) {
    cout<<"Before adding value inside function num="<<num<<endl;
    num=num+100;
     cout<<"After adding value inside function num="<<num<<endl;
}
int main() {
    int x=100;
    cout<<"Before function call x="<< x<<endl;
    change(x);//passing value in function
    cout<<"After function call x="<< x<<endl;
return 0;
}
```

The code produces the following output

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=200
```

Passing pointer as argument in function

Pointer as a function parameter is used to hold addresses of arguments passed during function call. This is also known as **call by reference**. When a function is called by reference any change made to the reference variable will effect the original variable.



Note:

A function can also return a pointer to the calling function. In this case you must be careful, because local variables of function doesn't live outside the function. They have scope only inside the function. Hence if you return a pointer connected to a local variable, that pointer will be pointing to nothing when the function ends.

Example#8

Make a function swap and pass two pointer type integer arguments and swap the value of both arguments.

Solution

```cpp
#include <iostream>
using namespace std;
void swap(int *n1, int *n2);

int main()
{
    int num1 = 5, num2 = 10;

    // address of num1 and num2 is passed
    swap( &num1, &num2);

    cout<<"num1 ="<<num1;
    cout<<"num2 ="<<num2;
    return 0;
}

void swap(int* n1, int* n2)
{
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

The code produces the following output

```
num1 =10num2 =5
```

Example#9

Write a program that declare integer variable and initialize it with 10. Assign the address of that variable to a pointer type integer variable. Make function named as Addone pass pointer variable into function and update the value of integer variable.

Solution

```
#include <iostream>
using namespace std;
void addOne(int* ptr) {
  (*ptr)++; // adding 1 to *ptr
}

int main()
{
  int* p, i = 10;
  p = &i;
  addOne(p);

  cout<<*p; // 11
  return 0;
}
```

The code produces the following output

```
11
```

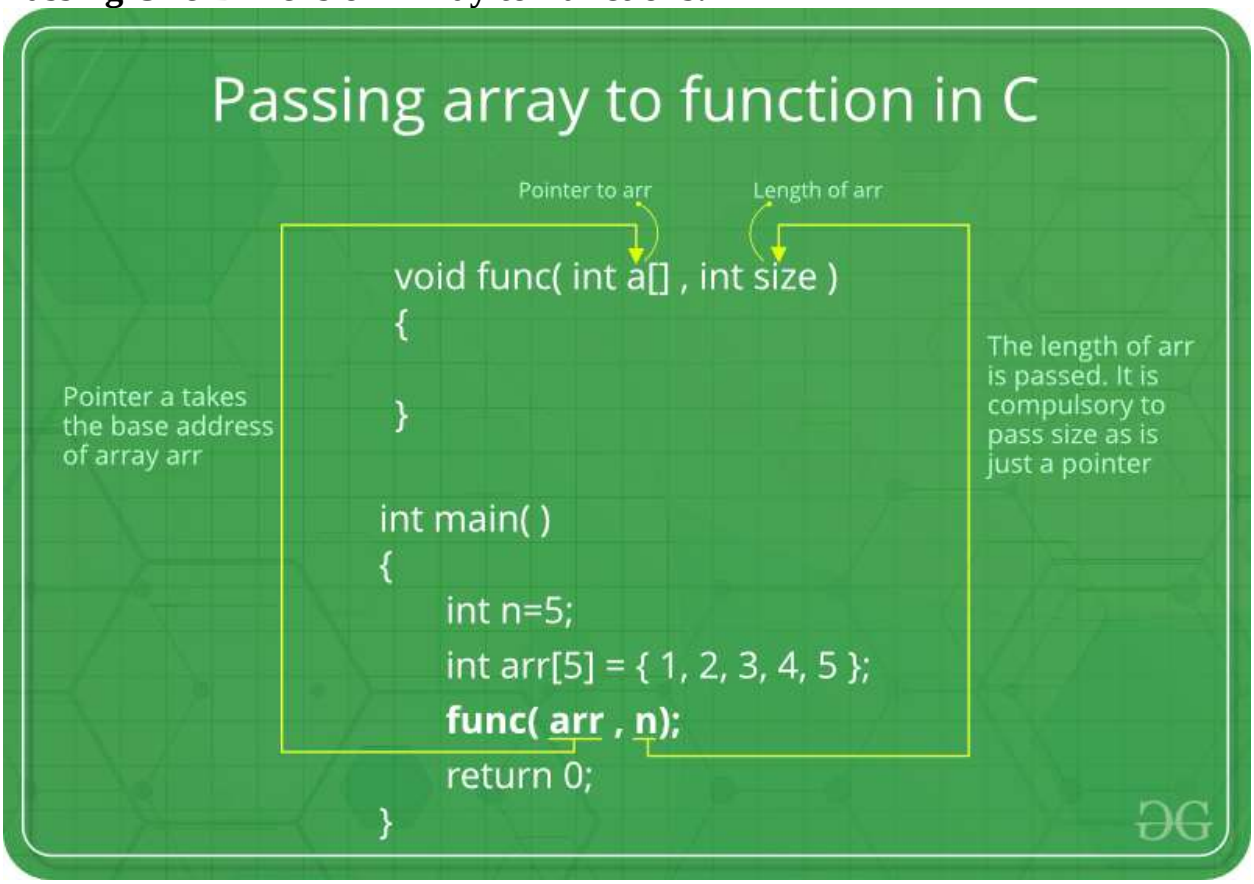
Example#10

Write a function named as square pass pointer type integer variable as an argument and change return the square of the integer variable without using return keyword.

Solution
``#include <iostream>
using namespace std;
int x=2;
void square(int* n) {
  *n=*n * *n;
}

int main()
{
 square(&x);
 cout<<x;
}``` |
| **The code produces the following output** |
|  |

Passing One-Dimension Array to Functions:



C++ does not allow to pass an entire array as an argument to a function. However, You can pass a pointer to an array by specifying the array's name without an index.

If you want to pass a single-dimension array as an argument in a function, you would have to declare function formal parameter in one of following three ways and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received.

Way-1

Formal parameters as a pointer as follows −

```
void myFunction(int *param) {
   .
   .
   .
}
```

Way-2

Formal parameters as a sized array as follows −

```
void myFunction(int param[10]) {
   .
   .
   .
}
```

Way-3

Formal parameters as an unsized array as follows −

```
void myFunction(int param[]) {
   .
   .
   .
}
```

Now, consider the following function, which will take an array as an argument along with another argument and based on the passed arguments, it will return average of the numbers passed through the array as follows:

Example#11

Write a function named as getAverage() pass two arguments of one array and other size of array. Function getAverage() should calculate sum and average of that all elements of array.

Solution

```
double getAverage(int arr[], int size) {
  int i, sum = 0;
  double avg;

   for (i = 0; i < size; ++i) {
      sum += arr[i];
   }
   avg = double(sum) / size;

   return avg;
}
#include <iostream>
using namespace std;

// function declaration:
double getAverage(int arr[], int size);

int main () {
   // an int array with 5 elements.
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   // pass pointer to the array as an argument.
   avg = getAverage( balance, 5 ) ;

   // output the returned value
   cout << "Average value is: " << avg << endl;

   return 0;
}
```

The code produces the following output

```
Average value is: 214.4
```

Passing array to function using call by reference

Solution

```cpp
#include <iostream>
using namespace std;
void disp( int *num)
{
    cout<<*num;
}

int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    for (int i=0; i<10; i++)
    {
        /* Passing addresses of array elements*/
        disp (&arr[i]);
    }

    return 0;
}
```

The code produces the following output

```
1234567890
```

Pass 2D array as a parameter

A one dimensional array can be easily passed as a pointer, but syntax for passing a 2D array to a function can be difficult to remember. One important thing for passing multidimensional arrays is, first array dimension does not have to be specified. The second (and any subsequent) dimensions must be given

1) When both dimensions are available globally (as a global constant).

Solution

```
#include <iostream>
using namespace std;
const int M = 3;
const int N = 3;

void print(int arr[M][N])
{
    int i, j;
    for (i = 0; i < M; i++)
      for (j = 0; j < N; j++)
        cout<<arr[i][j];
}
 int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr);
    return 0;
}
```

The code produces the following output

```
123456789
```

2) When only second dimension is available globally (either as a macro or as a global constant).

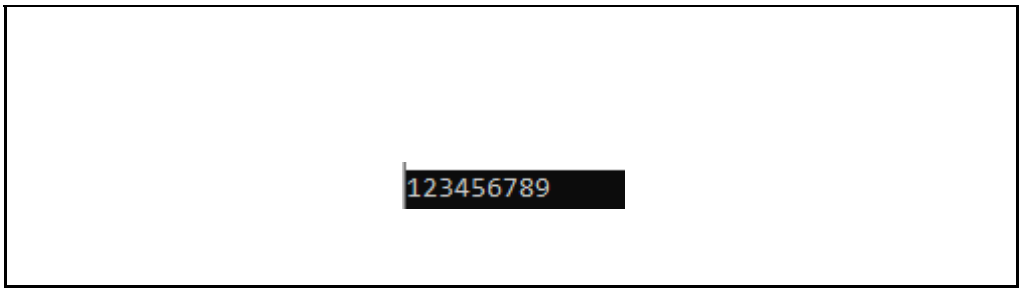
Solution

```
#include <iostream>
using namespace std;

const int N = 3;

void print(int arr[][N],int m)
{
    int i, j;
    for (i = 0; i < m; i++)
      for (j = 0; j < N; j++)
        cout<<arr[i][j];
}
 int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr,3);
    return 0;
}
```

The code produces the following output

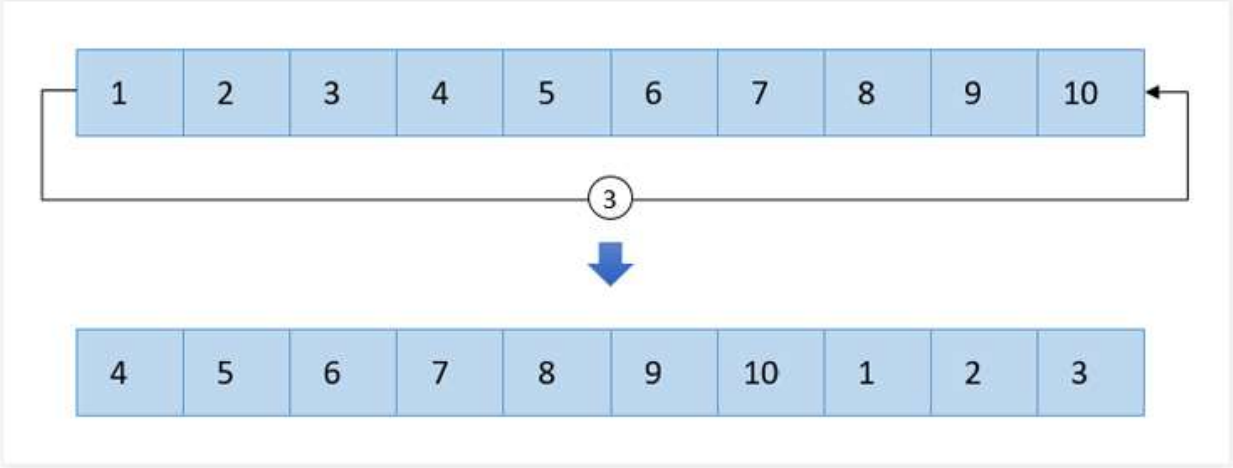


Challenge#1Write a program to print a number which is entered from keyboard using pointer.

Challenge#2: Write a function which will take pointer and display the number on screen. Take number from user and print it on screen using that function.

Challenge#3 Write a program that pass two array into function and merge those two array into third array.

Challenge#4: write a program that pass an array into function and left rotate the array by n position. Here as example given below



Challenge#5: write a Function that checks whether the 3X3 matrix is sparse matrix or not, matrix should be pass to the function.

Sparse matrix?

Sparse matrix is a special matrix with most of its elements are zero. We can also assume that if (m * n) / 2 elements are zero then it is a sparse matrix.